

《网络通信中的多线程编程》

作者：王琦

创建时间：2016年7月3日

最后修改：2016年7月3日

摘要：本文从 suri 项目的可维护性出发，阐述并解释了 suri 项目网络通信模块中的多线程机制。

1. 为什么 ServerCommunicator 需要两个线程

ServerCommunicator 主要承担两件工作：

- 在 ServerWidget 发生鼠标事件，MVP矩阵或鼠标信息被更新的时候，发送消息给 UI left/middle/right.
- 接受 UI left 发送来的消息，将其转发给 UI middle

1.1 由异步 I/O(Boost.asio) 实现的 ServerCommunicator

引用昊南师兄的话

“

事件驱动+异步编程+非阻塞IO，低系统资源消耗下的高性能
单个线程就可以同时向所有client发消息。

基于上述原因，我们采用异步 I/O 实现了 ServerCommunicator

1.2 基于异步 I/O 的 ServerCommunicator 多线程实现

由于 ServerCommunicator 需要从事收、发两件工作：

- 发送工作需要通过条件变量等待消息队列中的消息。当消息队列为空时，线程将被挂起。
- 由于我们希望发送线程被挂起的情况下，接收工作仍不受影响。

因此，我们为 ServerCommunicator 引入了多线程机制：

```

void ServerCommunicator::communicate() {
    ioService_.post(std::bind(&ServerCommunicator::startAccept, this));
    for (int i = 1; i <= 2; ++ i) {
        ioThreads_.push_back(std::thread([this] {
            ioService_.run();
        }));
    }
}

```

在两个线程中执行 ioService 任务池中的工作。

在完成所有 socket 连接后，将发送(startSend)和接收(startRecv)压入任务池中：

```

void ServerCommunicator::handleAccept(
SocketPointer socketPtr, const boost::system::error_code &error) {
    assert(! error);
    std::unique_lock<std::mutex> socketLck(socketsMtx_);
    sockets_.push_back(socketPtr);
    if (sockets_.size() != EXPECTED_SOCKET_SZ_) {
        startAccept();
    } else {
        ioService_.post(std::bind(&ServerCommunicator::startSend, this));
        ioService_.post(std::bind(&ServerCommunicator::startRecv, this));
    }
}

```